Turban-verkkosovelluksen käyttöliittymän suunnittelu ja toteutus

Viestinnän koulutusohjelma
Verkkoviestintä
Opinnäytetyö
21.8.2010

Mikko Lakomaa

Metropolia

Metropolia

Kulttuurialat

TIIVISTELMÄSIVU

Koulutusohjelma	Suuntautumisvaihtoehto
Viestinnän koulutusohjelma	Verkkoviestintä

Tekijä
Mikko Lakomaa

Työn nimi
Turban-verkkosovelluksen käyttöliittymän suunnittelu ja toteutus

Työn ohjaaja/ohjaajat
Matti Rantala

Työn laji	Aika	Numeroidut sivut + liitteiden sivut
Opinnäytetyö	21.8.2010	33

TIIVISTELMÄ

Opinnäytetyö käsittelee Turban-verkkosovelluksen suunnittelua ja toteutusta. Siinä tutustutaan verkkosovelluksen rakentamiseen liittyviin haasteisiin sekä käyttöliittymäsuunnitteluun. Työn lähestymistapana on esitellä käyttöliittymän eri elementteja ja niiden rakentamiseen käytettyjä käytettävyyden menetelmiä sekä ihmisen havaintokykyä hyödyntäviä seikkoja. Työssä tutustutaan myös verkkosovelluksen rakentamisen tekniseen puoleen.

Työ on hankkeistettu Art and Design City Helsinki Oy:lle, jossa kirjoittaja on työskennellyt useamman vuoden ajan sisältösuunnittelijana ja vastannut muun muassa yrityksen Urbaanimedianäyttöjen hallinnasta. Turban verkkosovellus on edellä mainittujen näyttöjen sisällönhallintaan luotu ohjelma, jonka tavoitteena on yhtenäistää ja helpottaa näyttöjen hallintaa.

Työn tuloksena on verkkosovelluksen lisäksi reflektointia työn toteutuksen aikana heränneistä kysymyksistä ja tehtyjen ratkaisujen onnistumisista ja epäonnistumisista sekä parannus- ja kehitysehdotuksia. Muun muassa Urbaanimedian soitto-ohjelma olisi hyvä tehdä kokonaan uudelta pohjalta ja Turbaniin voitaisiin lisätä tuki kosketusnäyttöjä käyttäville laitteille.

Teos/Esitys/Produktio
Turban-verkkosovellus

Säilytyspaikka
Aralis-kirjastokeskus

Avainsanat
Verkkosovellus, käyttöliittymät, sisällönhallinta, AJAX, jQuery, WWW

Metropolia

Culture

Degree Programme in	Specialisation
Media	New Media Design

Author
Mikko Lakomaa

Title
The Planning and Design of the User Interface in the Turban Web Application

Tutor(s)
Matti Rantala

Type of Work	Date	Number of pages + appendices
Bachelor´s Thesis	21st August 2010	33

The topic of this thesis is planning and designing the user interface in the Turban web application. The goal is to explore the user interface design and the challenges present when developing web applications. This is carried out by looking at different user interface elements and the usability and human cognition methods used when building them. The present thesis also looks into the technical side of web application development.

The thesis has been conducted in collaboration with Art and Design City Helsinki Ltd. For several years the author has been working as a content developer in the company and has been responsible for the management of the Urbanmedia screens. The Turban web application was created to manage the afore-mentioned screens. The goals for Turban's development were to create a program that allows easier and more unified control and content management of the Urbanmedia screens.

The end result included a web application as well as questions, solutions and revelations that came up during this project. These include successes and failures but also suggestions for improving and updating both Turban and Urbanmedia. For example completely overhauling the Urbanmedia player software would be a good idea and in the future Turban should be improved to better support devices employing touchscreens.

Work / Performance / Project
Turban web application

Place of Storage
Aralis Library and Information Centre

Keywords
Web application, user interfaces, content management, AJAX, jQuery, WWW

SISÄLLYS

1 JOHDANTO 2
2 MIKÄ IHMEEN URBAANIMEDIA? 2
2.1 Urbaanimedian tekniset tiedot 4
2.2 Urbaanimedian hallinta ja turban-projektin haasteet 4
3 ENTER THE TURBAN! 5
4 TURBANIN KÄYTTÖLIITTYMÄN ELEMENTIT JA OMINAISUUDET 7
4.1 Drag-and-drop 9
4.2 Mediakirjasto 11
4.3 Roskakori 12
4.4 Viestijärjestelmä 13
4.5 Tallentaminen 15
4.6 Muokkaaminen 16
4.7 Opasteet 17
5 TOTEUTUS 19
5.1 Selaintuki 19
5.2 AJAX 20
5.3 jQuery 21
5.4 Tietokanta 21
5.5 Tietoturva 23
6 ULKOASUN KEHITYSKAARI 23
6.1 Napit 24
6.2 Välilehdet 26
7 MITEN TURBANIA VOISI KEHITTÄÄ ETEENPÄIN? 28
8 YHTEENVETO 29
LÄHTEET 32

1 JOHDANTO

Toiminnallinen opinnäytetyöni koostuu kirjallisesta osasta ja teososasta. Tutkin yhden tapauksen avulla, miten verkossa toimivan sovelluksen (web application) käyttöliittymä syntyy ja millaisia haasteita sellaisen kehittäjällä on edessään. Kuinka ihmisen havaintokyvyn toimintaa, opittuja konventioita ja psykologiaa voidaan hyödyntää käyttöliittymän rakentamisessa sekä hyvän käytettävyyden tavoittelemista verkkosovelluksen käyttöliittymäsuunnittelussa. Lisäksi paneudun kyseessä olevan verkkosovelluksen vaatimiin teknisiin ratkaisuihin.

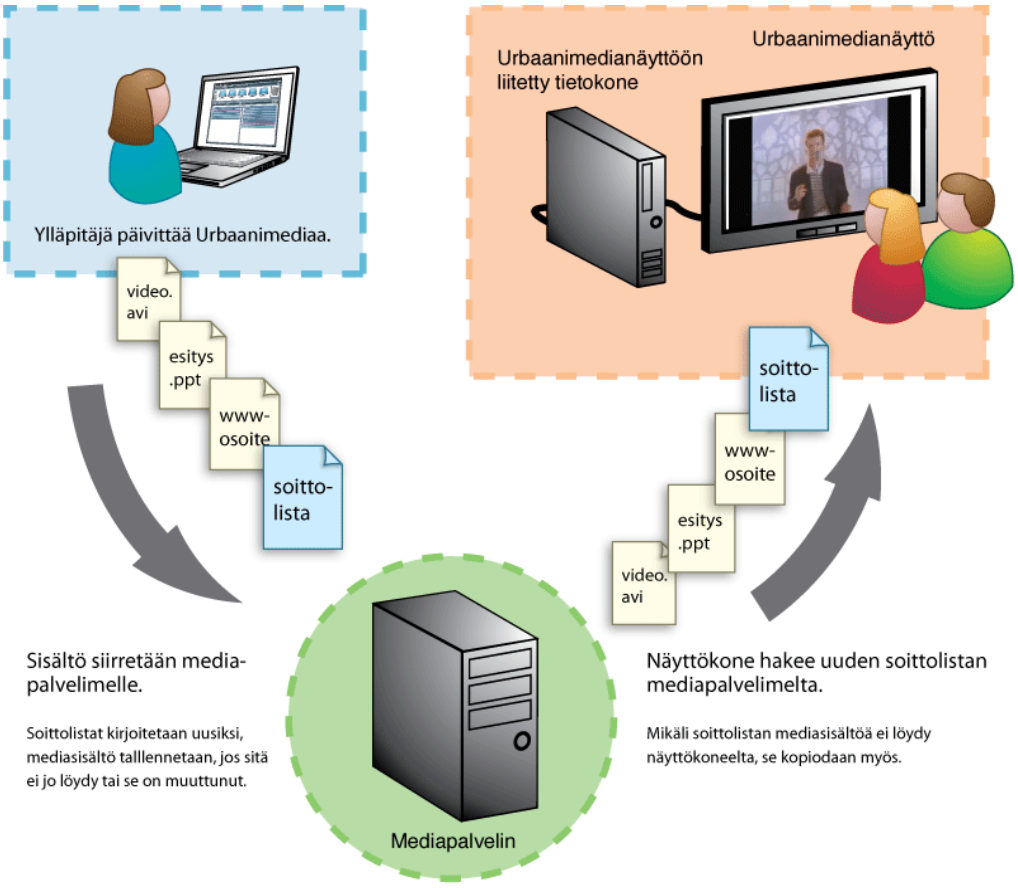
Teososa koostuu Art and Design City Helsinki Oy:lle (ADC) rakentamastani Urbaanimedia-näyttöjen soittolistoja muokkaavasta verkkosovelluksesta, nimeltään Turban - Urbaanimedia soittolistaeditori. Sillä hallitaan Urbaanimedia-näytöillä näkyvää materiaalia (videot, powerpointit, verkkosivut). Turban sisällönhallintasovelluksen suunnittelussa ja toteutuksessa on pyritty siihen, että sovelluksen käyttäminen onnistuisi mahdollisimman intuitiivisesti varsin vähäisellä opettelulla. Tähän tavoitteeseen on pyritty muun muassa hyödyntämällä Microsoft Windows ja Apple Mac OSX käyttöjärjestelmistä tuttuja drag-and-drop-, roskakori- ynnä muita paradigmoja verkkosovelluksen maailmassa.

2 MIKÄ IHMEEN URBAANIMEDIA?

Urbaanimedianäytöt ovat Arabianrannassa useassa eri paikassa (Arabian kauppakeskus, useimmat alueen oppilaitokset) sijaitsevia teräväpiirtotelevisioita, joihin on jokaiseen liitetty tietokone. Niillä voidaan esittää muun muassa mainoksia, ajankohtaista tietoa alueen tapahtumista, opiskelijoiden teoksia ynnä muuta sellaista. Järjestelmä käyttää Back Office Services Oy:n (BOS) kehittämää Windowsissa toimivaa ohjelmistoa, joka pystyy näyttämään videoita, powerpoint-esityksiä ja verkkosivuja kullekkin näytölle asetetun soittolistan mukaisesti. Kukin näyttökone lataa näytöillä

esitettävän sisällön ja soittolistat internetin välityksellä erilliseltä mediapalvelimelta, jonka kautta hoituu myöskin sisällön ja soittolistojen päivitys.

Urbaanimedianäyttöjen käyttämä ohjelmisto toimii siten, että sen asetuksiin määritetään jokin tietty kansio (esimerkiksi “screens\kauppakeskus”) mediapalvelimella, ja tietyin väliajoin ohjelmisto ottaa yhteyden mediapalvelimeen, tarkistaa onko soittolista päivittynyt (mediapalvelimella näytön kansiossa sijaitsevaan asetustiedostoon on merkitty uudempi päivämäärä ja/tai aika) ja mikäli näin on, ohjelmisto lataa uuden soittolistan ja sisällön näytön koneelle ja alkaa soittaa soittolistaa läpi. (Kuva 1.)



Kuva 1. Urbaanimedian toimintaperiaate.

2.1 Urbaanimedian tekniset tiedot

Urbaanimedianäyttöjen ohjelmisto toimii tällä hetkellä Windows XP:ssa ja se käyttää Windows Media Playerin, PowerPoint Viewerin ja Internet Explorerin ohjelmistorajapintoja sisällön näyttämiseen. Urbaanimedianäyttöjen tietokoneille on asennettu ffdshow-koodekkipaketti, jonka avulla voidaan soittaa useimpia video- ja ääniformaatteja. Koska Urbaanimedianäytöt ovat 720p tarkkuudella toimivia teräväpiirtotelevisioita, maksimitarkkuus mediasisällölle on 1280x720 pikseliä. Powerpoint-sisällön kohdalla taas on kyse PowerPoint Viewerin versiosta. Aiemmin Urbaanimedianäyttöihin oli asennettu vanha versio PowerPoint Viewerista, joten kaikkien powerpoint-esitysten piti olla PowerPoint ’95 muodossa. Nyt tätä ongelmaa ei ole, sillä Turbanin rakentamisen yhteydessä näyttökoneisiin päivitettiin uusimmat versiot ohjelmista.

Samalla myös Urbaanimedianäyttöjen pääsyä verkkoon rajoitettiin niin, että ne pystyvät kommunikoimaan vain mediapalvelimen kanssa. Myös kuhunkin koneeseen asennetusta käyttöjärjestelmän ja ohjelmien yhdistelmästä tehtiin yhtenäinen, jolloin jonkin koneen hajotessa voidaan asentaa sama ohjelmisto- ja käyttöjärjestelmäkokonaisuus helposti uudelleen levykuvan avulla.

2.2 Urbaanimedian hallinta ja turban-projektin haasteet

Urbaanimedianäyttöjen sisällön hallintaan ei ollut varsinaisia omia työkaluja. Alunperin tarkoitus oli ilmeisesti, että hallinta tapahtuisi Urbaanimedianäyttöjen ohjelmiston kautta, mutta se ei soveltunut lainkaan usean näytön hallintaan ja oli muutenkin kömpelö. Päivittäminen oli jossain määrin helpompi tehdä lataamalla FTP-ohjelmalla Urbaanimedianäyttöjen mediapalvelimelta soittolistat sisältävät tekstitiedostot ja muokkaamalla niitä tekstieditorilla.

Hallintatyökalujen puutteesta syntyi kolme erilaista ongelmaa:

Päivitys oli työlästä: Ensin joutui lataamaan soittolistat mediapalvelimelta, sitten muokkaamaan niitä tekstieditorilla ja lähettämään uudet soittolistat takaisin mediapalvelimelle. Tämän lisäksi käyttäjän piti muistaa tallentaa mahdolliset mediatiedostot jokaisen näytön kansioon ja muistaa päivittää kunkin soittolistakansion toisesta tekstitiedostosta päivämäärä, jotta Urbaanimedianäytöt ymmärtävät hakea uudet soittolistat mediapalvelimelta. Välillä myös esimerkiksi powerpoint-tiedostoja

joutui muuttamaan vanhempaan formaattiin, jotta ne toimivat Urbaanimedian soitto-ohjelmassa.

Systeemin käytön opettaminen muun muassa uusille harjoittelijoille vei reilusti aikaa. Kyseessä oli kuin vanhan heimoperinteen mukainen prosessi, jossa edellinen harjoittelija yleensä opasti seuraavaa, jolta sitten tieto Urbaanimedian päivittämisestä siirtyi jälleen eteenpäin. Toisin sanoen jos tämä tiedonjakamisen ketju jostain syystä katkeaisi (esimerkiksi uutta harjoittelijaa ei saada ennen kuin edellinen lopettaa), olisi mahdollista, että kukaan ei olisi enää oikein perillä Urbaanimedian käytöstä.

Näytöille unohtui monesti esimerkiksi mainoksia tapahtumista, joita olisi pitänyt esittää vain tietyn ajan. Kun tapahtumat olivat päättyneet, kukaan ei enää muistanut poistaa mainoksia näytöiltä. Katsojat tai ne tahot, jotka olivat alun perin näytettävän materiaalin lähettäneet, huomasivat nämä vanhat mainokset monesti ennen Urbaanimedian ylläpitäjiä ja ilmoittivat sitten tyytymättömyydestään ADC:lle.

3 ENTER THE TURBAN!

Aloin suunnitella ratkaisua edellä mainittuihin ongelmiin. Jo alusta lähtien oli selvää, että ratkaisu olisi verkkosovellus, sillä sellaisten kehittäminen on minulle työpöytäohjelmia tutumpaa, johtuen verkkoviestinnän koulutusohjelmassa saamastani opetuksesta. Verkkosovellukset ovat ohjelmia, joita käytetään WWW-selaimella web-käyttöliittymien kautta internetissä tai intranetissä. Verkkosovelluksia ovat muun muassa web-sähköpostit (esim. Gmail), verkossa toimivat tekstinkäsittelyohjelmat (Google Documents, Bespin ym.) tai muut, usein työpöytäohjelmista verkkoon siirretyt sovellukset (esim. Google Calendar). Toisin kuin perinteisiä työpöytäsovelluksia, verkkosovelluksia voidaan käyttää millä tahansa käyttöjärjestelmällä, kunhan siinä on asennettuna riittävän hyvä WWW-selain. Haittapuolena verkkosovelluksissa taasen on riippuvaisuus verkkoyhteydestä. Koska Turbanissa tätä tarvitaan joka tapauksessa sisällön siirtämiseen, ei tämä luonnollisesti ollut ongelma vaan pikemminkin elinehto.

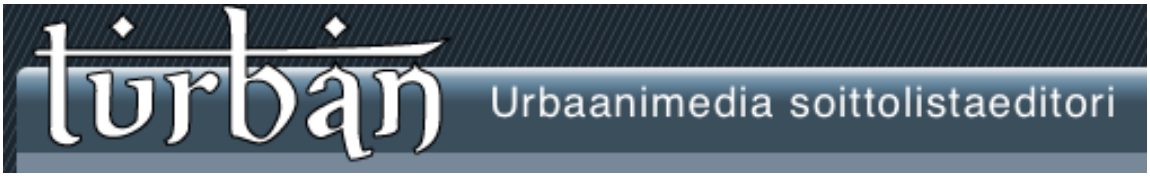
Teknisesti verkkosovelluksien kehittämiseen käytetään samoja kieliä ja menetelmiä (JavaScript, PHP, HTML, CSSRuby on Rails, Java ym.) kuin staattisten verkkosivustojenkin kehittämiseen. Monet verkkosovellukset käyttävät yhden ruudun näkymää, eli toisin kuin perinteisissä verkkosivustoissa itse sivua ei vaihdeta vaan sen sisältöä päivitetään taustalla edellä mainittujen tekniikoiden avulla. Näin verkkosovelluksen toiminto on lähempänä työpöytäsovelluksen toimintaa:

käyttöliittymä pysyy yhtenäisenä alusta loppuun eikä välillä vaihdu täysin. Tällaista ohjelmistorakennetta hyödynnetään myös Turbanissa. (Wikipedia 2009, Web application.)

Ero työpöytäsovellusten ja verkkosovellusten välillä on yhä häilyvämpi, sillä monet verkkosovellukset muistuttavat ulkonäöltäänkin jonkin tietyn käyttöjärjestelmän sovelluksia (esimerkiksi 280 Slides on verkkosovelluksena toteutettu kopio Applen Keynote-työpöytäsovelluksesta) ja sivukohtaisten selainten (Fluid, Google Chrome, Mozilla Prism) avulla voidaan luoda verkkosovelluksista kuvakkeet tietokoneen työpöydälle, jolloin verkkosovellus toimii päällisin puolin hyvin samankaltaisesti kuin työpöytäsovelluskin. Näin siis Turbaniakin olisi mahdollista käyttää työpöytäsovelluksen tapaan.

Turbanin tavoitteena oli ratkaista kaikki Urbaanimedian hallinta -luvussa mainitut pääongelmat ja lisäksi luoda systeemi, joka suorittaa tiettyjä toimintoja automaattisesti ja joka voidaan oppia helposti. Käyttöön ei tarvita minun opastustani vaan esimerkiksi joku toinen työntekijä voi tehdä opastuksen tarvittaessa. Vaihtoehtoisesti uusi työntekijä voidaan perehdyttää Urbaanimedianäyttöjen hallintaan antamalla hänen tutustua Turbanin sisältä löytyvään opastukseen. Pähkinänkuoressa Turbanin siis pitää olla helppokäyttöinen, osittain automatisoitu ja vakaa.

Turban-nimen keksin vasta projektin loppuvaiheilla. Koska Urbaanimedianäytöt löytyvät tätä kirjoittaessa pelkästään Arabianrannan alueelta, Turban-nimessä yhdistyy arabimaissa tyypillinen turbaanihattu ja Urbaanimedia. Samalla nimi symboloi ikään kuin päätä (ylläpitäjä), joka hallitsee ruumista eli mediapalvelinta ja näyttöjä. Arabimaihin viittaa myöskin ohjelman logo (Kuva 2).



Kuva 2. Turbanin logo.

Päivittämisen hankaluuteen löytyi lääke helpoiten, sillä jo koko päivitysprosessin keskittäminen yhdessä selainikkunassa tapahtuvaksi karsii pois pahimmat ongelmat eli tekstitiedostojen muokkaamisen, niiden kaivamisen kansioista ja lähettelemisen FTP:lla edestakaisin. Vanhan sisällön poistaminen soittolistoilla automaattisesti oli myöskin

ideatasolla helppo ratkaista arkipäiväisellä konventiolla: otetaan ruokakaupan lihapaketeista tuttu “parasta ennen”-päivämäärä ja liitetään se verkossa olevaan mediasisältöön. Kun mediasisällölle asetettu päivämäärä on ohitettu, heitetään “pilaantuneet” mediasisällöt pois soittolistoilta.

Yksi suurimpia haasteita oli kuitenkin verkkosovelluksen käytettävyys. Kuinka saisin siitä niin järkevän, että käyttäjällä ei tulisi ylitsepääsemättömiä ongelmia sovelluksen opettelussa ja käytössä? Suunnittelijalla ja käyttäjällä on kuitenkin varsin eriävät lähtökohdat verkkosovelluksen ymmärtämiseen. Siinä missä suunnittelijan mentaalinen malli perustuu loogisiin rakenteisiin ja absoluuttiseen ymmärrykseen toteutettavasta verkkopalvelusta, käyttäjällä puolestaan verkkopalvelun malli syntyy aiemmasta kokemuksesta kumpuavien uskomusten ja sattumien pohjalle. Suunnittelijan mallin välittäminen käyttäjälle on usein erittäin hankalaa. Jotta näiden kahden mentaalisen mallin välille löydettäisiin sopiva kompromissi, on hyvä ottaa mallia muista sovelluksista, niiden ulkonäöstä ja käytettävyyskonventioista, sillä käyttäjä on jo sisäistänyt niiden toiminnan aiemman kokemuksen pohjalta. Kuitenkin myös käyttöliittymän yhtenäisyys itsensä kanssa on tärkeää. Käyttöliittymän elementit eivät saisi vaihdella miten sattuu, vaan samanlaisten toimintojen suorittaminen pitäisi onnistua samanlaisen prosessin avulla ja verkkosovelluksessa tai verkkosivustolla tulisi olla joka paikassa yhtenäinen ulkonäkö, jotta käyttäjä kokee olevansa saman käyttöliittymän ääressä. (Parkkinen 2002, 70–74; Garrett 2003, 116-119.)

4 TURBANIN KÄYTTÖLIITTYMÄN ELEMENTIT JA OMINAISUUDET

Kuvassa 3 on Turbanin pääkäyttöliittymä. Se koostuu kolmesta osiosta: yläreunassa on Urbaanimedianäyttöjä kuvaavat ikonit omassa osiossaan, ja näitä klikkaamalla voidaan ladata kunkin näytön soittolista näyttöosion alla olevaan soittolistaosioon. Soittolistaosion vieressä on viimeinen osio, mediakirjasto, jossa säilytetään kaikkea Urbaanimedian mediapalvelimelle tallennettua sisältöä.

Lisäksi näyttöosion vierestä löytyy roskakori, jonka avulla voidaan poistaa näyttöjä ja mediasisältöä Turbanista. Soittolista- ja mediakirjasto-osioiden yläreunassa on nimipalkki, jonka oikeasta reunasta löytyy osioon liittyvien toimintojen napit. Näyttöosiossa nämä on pystysuuntaisen tilan säästämisen vuoksi laitettu oikeaan reunaan, sillä useimmissa nykyaikaisissa tietokonenäytöissä vaakaresoluutio on

suurempi kuin pystyresoluutio. Vaikka tämä rikkookin käyttöliittymän yhtenäisyyttä, se myös vähentää pystysuuntaisen vierityksen tarvetta.

Yksi Turbanin kaltaisen yhden näkymän verkkosovelluksen rakentamisen haasteista onkin valita näkymälle sopiva koko. Itse käytin tämän pohjana pienintä suosittua ruutukokoa, eli 1024x600 pikselin netbook-kannettavien resoluutioita. Ottamalla vierityspalkit sivulta pois valitsin Turbanin käyttöliittymän leveydeksi noin 900 pikseliä. Pystyresoluution kannalta en ottanut huomioon muuta kuin sen, että kaikki pääelementit mahtuvat ruudulle samalla kerralla ja näin käyttäjä näkee yhdellä silmäyksellä mitä kaikkea hän voi käyttää. Jakob Nielsenin (2010) mukaan verkkosivun kävijälle tärkeimpien elementtien tulisi olla “foldin” eli vasta postiluukusta kolahtaneen sanomalehden taitetun etusivun yläosassa. Verkkosivustoissa tämä tarkoittaa käytännössä sivun yläreunaa, joka nähdään ensimmäiseksi verkkosivustolle saavuttaessa. Sen kohdan, johon edellä mainittu “taite” tulee, määräävät selainikkunan koko ja näyttöruudun resoluutio. Fiz Yazdi ja Joe Leech (2009), kuten myös Nielsen, kuitenkin muistuttavat, että ihmiset ovat oppineet rullaamaan verkkosivuja alaspäin, joten kaikkea ei ole tarpeellista ahtaa “taitteen” yläpuolelle. Jardi ja Leech mainitsevat myös, että käyttöliittymäsuunnittelun avulla voi käyttäjälle antaa vinkkejä siitä, että sivu jatkuu alaspäin ja näin houkutella häntä rullaamaan sivua. Turbanissa tämä tapahtuu käytännössä silloin, kun mediakirjastossa on niin paljon sisältöä, että se ei mahdu ruudulle kerralla. Tällöin osittain näkyvät mediasisällöt ja sivulla oleva rullauspalkki kertovat sivun jatkuvan.

Kuva 3. Turbanin käyttöliittymä.

4.1 Drag-and-drop

Drag-and-drop, eli suomeksi raahaa ja tiputa, on tällä hetkellä käytössä kaikissa suosituimmissa tietokoneiden käyttöjärjestelmissä kuin myös joissakin älypuhelimien käyttöjärjestelmissä. Yksinkertaisuudessaan se antaa mahdollisuuden raahata asioita paikasta toiseen, raahaamisen ja/tai tiputtamisen aiheuttaessa jonkinlaisen toiminnon. Koska ihmiset ovat oppineet jo tietokoneita käyttäessään raahailemaan tiedostoja ym. paikasta toiseen, miksei tätä voisi käyttää myös verkkosovelluksen puolella? (Wikipedia 2010, Drag and drop.)

Turbanissa drag-and-dropia käytetään mediasisällön järjestämiseen soittolistoissa ja siirtämiseen soittolistan ja mediakirjaston (kuva 4) sekä roskakorin välillä. Vastaavasti näyttöjä voi siirtää roskakoriin poistettavaksi. Perusparadigma muistuttaa hyvin paljon tiedostojen siirtämistä kansiosta toiseen tai musiikinsoitto-ohjelman (iTunes, foobar2000, Winamp ym) soittolistan järjestämistä, niin teoriassa kuin käytännössäkin.

Haastavan drag-and-dropista verkkosovelluksen yhteydessä tekee se, että tyypillisesti verkkosivuilla ja -sovelluksissa on totuttu vain klikkailemaan nappeja ja linkkejä, avaamaan valikoita yms. Drag-and-drop on siis verkkosovelluksissa vasta tekemässä tuloaan, esimerkiksi Google lisäsi drag-and-drop-toiminnallisuuden Gmail-sähköpostisovelluksen tunnisteiden lisäämiseen sähköpostiviesteihin vain vähän aikaa sitten. Toisin sanoen vaikka drag-and-drop on tuttu tapa toimia, käyttäjät eivät ehkä ymmärrä kokeilla sen käyttöä verkkosovelluksissa.

Jakob Nielsenin mukaan yksi drag-and-dropin ongelmista on joissain tapauksissa selkeän affordanssin puute. Affordanssilla tarkoitetaan jossain objektissa havaittavaa mahdollisuutta toiminnalle. Drag-and-dropin yhteydessä ei usein näe mitenkään, että jokin objekti on raahattavissa, joten objektista puuttuu nähtävä affordanssi. Vasta kun objektia yrittää raahata, huomaa, että raahaaminen on mahdollista. Yksi ratkaisu tähän ongelmaan on laittaa raahattavaan objektiin jonkinlainen ikoni. Esimerkiksi Applen iPhonen suosikkikontakteissa käytetään laatikkoa, jossa on kolme viivaa - ikään kuin liukunapin pintakuvio. Turbanissa drag-and-drop-affordanssia yritetään kuvata muuttamalla hiiren kursori käden näköiseksi siirto-kursoriksi, kun kursori viedään raahaamisen mahdollistavan kohteen päälle. (Nielsen 2008; Wikipedia 2010, Affordance.)

Kun Turbanissa raahataan joko näyttöä tai mediasisältöä, näyttää sovellus sen tulevan paikan, mikäli raahattava objekti on riittävän lähellä. Tämä indikaattori esitetään muuttamalla tiputuskohdan taustaväriä ja lisäämällä siihen raahattavan objektin muotoinen reunus. Tämä auttaa raahattavan objektin tiputtamisessa juuri oikeaan väliin, jolloin esimerkiksi soittolistaan laitettua mediasisältöä ei tarvitse enää uudelleen raahata haluttuun kohtaan soittolistassa. Indikaattorin avulla käyttäjä myös tietää, että hän on raahaamassa mediasisältöä tai näyttöä hyväksyttyyn paikkaan eikä yritä esimerkiksi tiputtaa näyttöä mediakirjastoon. Näin indikaattori toimii toisena drag-and-dropin affordanssina ja ilman sitä Turbanin drag&dropin käytöstä tulisi huomattavasti vähemmän intuitiivista.

Kuva 4. Drag and drop.

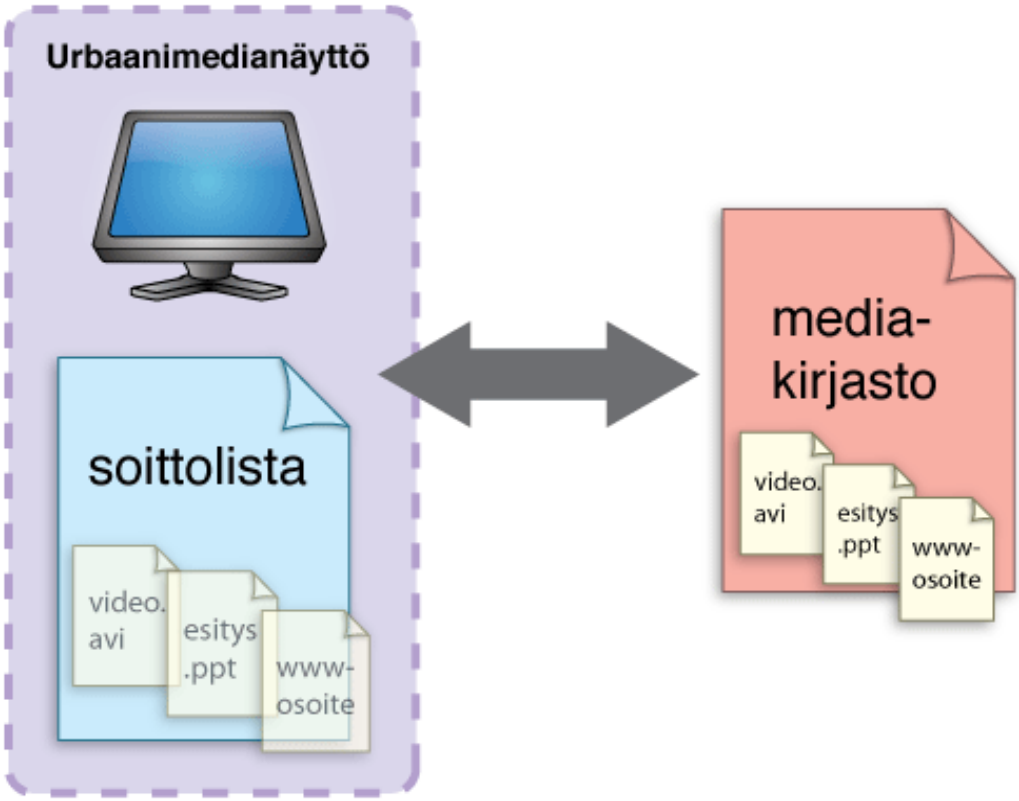
4.2 Mediakirjasto

Alun perin Urbaanimedian järjestelmässä ei ollut mediakirjastoa lainkaan, vaan se on pelkästään Turbania varten kehitetty ominaisuus. Ongelmana oli, että mediasisältö oli jaettuna ympäriinsä kunkin näytön kansioissa, sen mukaan mitä kunkin näytön soittolistalla näytettiin. Mediakirjasto sen sijaan pitää kaiken sisällön yhdessä kansiossa helposti saatavilla. Kun soittolista tallennetaan, kopioidaan mediakirjaston kansiosta tarvittavat tiedostot näytön kansioon, mikäli ne eivät jo ole siellä.

Mediasisällön yhteen paikkaan keskittämistä tärkeämpi mediakirjaston ominaisuus on kuitenkin se, että se luo pohjan koko Turbanin sisällönhallinnalle. Mediakirjastoon tallennettua sisältöä voidaan vapaasti siirtää drag-and-dropin avulla näyttöjen soittolistoihin ja pois niistä. Vastaavasti mediakirjaston avulla voidaan myös hallita mediasisältöä, joka ei ole millään soittolistalla. Tästä syystä se on olennainen osa Turbanin toiminnallisuutta.

Mediakirjaston malli on haettu pitkälti sellaisista musiikinsoitto-ohjelmista kuten iTunes, foobar2000, Winamp ym. Siihen kootaan kaikki mediasisältö, jota Urbaanimedianäytöillä voidaan näyttää. Musiikinsoitto-ohjelmista poiketen samaa mediasisältöä ei voi laittaa useampaa kertaa samalle soittolistalle. Päädyin tähän ratkaisuun, koska yleensä näyttöjen soittolistat ovat melko lyhyitä ja niin on myös itse sisältökin, joten soittolista toisto alkaa alusta melko nopeassa syklissä. Tämä samalla helpottaa mediakirjaston ja soittolistan hallintaa. Käyttäjän on helpompi ymmärtää mediasisällön siirtäminen paikasta toiseen, kun se voi olla vain joko soittolistalla tai sitten mediakirjastossa. Näin käyttäjä näkee helposti, mitkä mediakirjaston sisällöistä ovat käytössä ja mitkä eivät.

Mediakirjastossa kukin mediasisältötyyppi (video, powerpoint, verkko-osoite) on eroteltu käyttämällä erilaista taustaväriä: sininen videoille, vihreä powerpointeille ja violetti verkko-osoitteille. Tämä helpottaa eri sisältöjen löytämistä mediakirjastosta. (Kuva 5.)



Kuva 5. Turbanin sisällön rakenne. Jokaisella Urbaanimedianäytöllä on oma soittolista, johon siirretty mediasisältö viittaa mediakirjaston sisältämiin tiedostoihin ja osoitteisiin.

4.3 Roskakori

Roskakori on myös käyttöjärjestelmistä tuttu paradigma. Kun jotain halutaan poistaa, se viedään roskakoriin ja roskakori tyhjennetään. Turbanissa roskakoriin voi viedä näyttöjä ja mediasisältöä ja nämä voi myös palauttaa takaisin, jos roskakoria ei ole tyhjennetty. Kuvassa 6 nähdään, miten roskakori myös havainnollistaa, miten paljon sen sisällä on tavaraa: tyhjä roskakori, 1-3 mediasisältöä tai näyttöä = roskakori puolillaan, jos enemmän, roskakori on täynnä. Tämä ominaisuus on jälkeenpäin ajatellen hieman turha ja alan ymmärtää miksei vastaavaa löydy muista roskakori-implementaatioista. Oleellista ei nimittäin ole, kuinka paljon tavaraa roskakorissa on vaan onko siellä yleensä mitään.

Pääasiallinen syy roskakorille oli vähentää ruudulla olevien ikonien määrää. Jos sisällön poistaminen olisi toteutettu perinteisellä ”ruksi nurkassa” -menetelmällä, rukseja olisi monia kymmeniä kappaleita sisällön määrästä johtuen. Lisäksi tämä olisi hankaloittanut mediasisällön järjestämistä, sillä raahattava alue olisi ollut pienempi. Roskakori myös näppärästi estää käyttäjää poistamasta sisältöä vahingossa. Lisäksi siten saadaan tasan yksi paradigma sisällön poistamiseen: näytöt ja mediasisältö poistetaan samalla tavalla ja useammankin tiedoston poistaminen onnistuu yhdellä roskakorin tyhjennyksellä.

Turbanin koodissa on myös mahdollisuus roskakorin sisällön säilyttämiselle sessioiden välillä, mutta tämä kytkettiin pois, sillä yleensä sisältöä ei ole tarve poistaa kokonaan kuin harvoin ja silloin se yleensä poistetaan samalla. jos sisältö halutaan ottaa pois käytöstä, sen voi vain poistaa soittolistoilta. Tämä vähentää käyttöliittymän monimutkaisuutta, sillä käyttäjä näkee aina kaiken sisällön eikä joudu etsimään sitä roskakorista.

Kuva 6. Roskakorin eri tilat: Tyhjä, roskakorin sisältö alle 3 kpl, roskakorissa yli 3 kpl poistettavia.

4.4 Viestijärjestelmä

Koska verkkosovelluksen tulee myös antaa palautetta käyttäjän toimista, rakensin Turbaniin viestijärjestelmän. Viestityyppejä on kahdenlaisia: käyttäjän toimia vaativia ja pelkästään ilmoitusluontoisia. Käyttäjän toimia vaativia ovat esimerkiksi uuden näytön lisäämisen kaavake tai roskakorin tyhjentämisen hyväksyminen/peruminen.

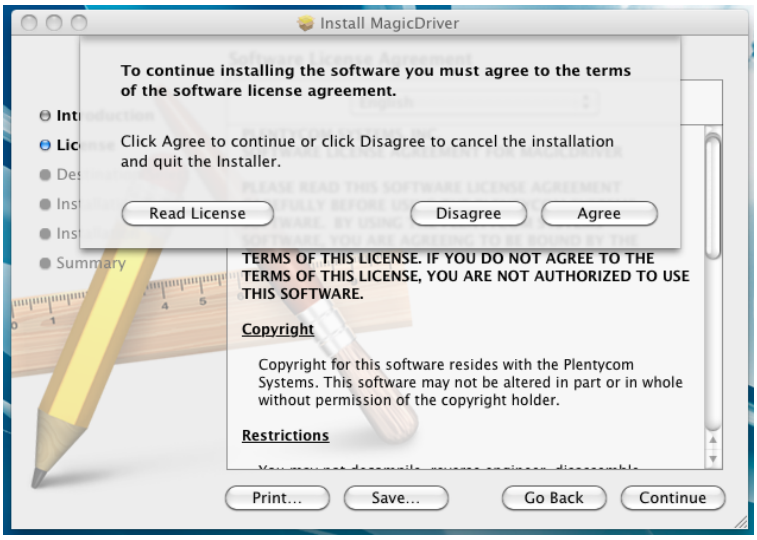
Ilmoitusluontoisia viestejä taas ovat muun muassa virheilmoitukset tai onnistuneen tallennuksen ilmoitus. Näin viestijärjestelmän tuottamat palautteet ovat kahdenlaisia: kognitiivisia eli käyttäjän huomion herättäviä, toiminnan toteutumista tai epäonnistumista kuvaavia viestejä sekä työnkulullisia viestejä, jotka kertovat, että käyttäjä lähestyy tavoitettaan tai on saavuttanut sen. Näiden viestien tuli olla huomiota herättäviä, muttei kuitenkaan sellaisia, että käyttäjän työ häiriintyy. (Parkkinen 2002, 135–136.)

Viestijärjestelmässä käytetään selainikkunan yläreunasta alas ponnahtavaa osiota (kuva 7). Näin viestit näkyvät aina yhdessä paikassa. Jos viestin näyttäminen kestää esimerkiksi tiedoston tallennuksen takia, näytetään latausanimaatiota, jonka aikana käyttäjä voi tehdä muita asioita. Ilmoitusluontoiset viestit sulkeutuvat itsestään viiden sekunnin kuluttua. Koska ne ovat yleensä korkeintaan parin rivin mittaisia, viestin ehtii hyvin lukea tuossa ajassa. Käyttäjän toimia vaativat viestit merkitään tummentamalla kaikki taustalla näkyvä, jotta huomio keskittyy pelkästään viestialueeseen (kuva 8). Nämä viestit joutuu sulkemaan käsin, joko “Sulje tämä viesti”-linkkiä klikkaamalla tai klikkaamalla tummennettua taustaa.

Viestijärjestelmää hyödynnetään kaikissa Turbanin kaavakkeita vaativissa toimissa, kuten tiedostojen tallentaminen tai näyttöjen lisääminen ja muokkaaminen. Tällä tavoin käyttöliittymä pysyy selkeämpänä eivätkä käyttöliittymän pääelementit vaihda paikkaa, sillä viestit tulevat aina sen päälle. Idea viestijärjestelmän ulkonäöstä tuli Mac OSX:n asennusohjelmista, joissa saattaa tulla ilmoituksia (esimerkiksi ilmoitus, että kone täytyy käynnistää uudelleen ohjelman asennuksen jälkeen) ohjelmaruudun yläreunasta ennen asennuksen alkamista (kuva 9).

Kuva 7. Viesti, joka ei vaadi käyttäjän toimia vaan sulkeutuu itsestään.

Kuva 8. Viestityyppi, joka vaatii käyttäjän toimia.



Kuva 9. Mac OSX:n asennusohjelman viesti.

4.5 Tallentaminen

Tallentamisen käsitteeseen Turbanissa kuuluu niin uuden mediasisällön tallentaminen, kuin myös uusien näyttöjen luonti ja niiden soittolistojen tallentaminen. Tämän toteuttamisessa pyrin yhtenäistämään käyttöä käyttämällä samoja ikoneita eri paikoissa yhdistämällä ne tiettyyn kontekstiin. Esimerkiksi lisäämistä kuvaava plus-ikoni on käytössä niin näyttöjen kuin mediakirjastonkin osiossa. Tämä konsistenssi takaa sen, että käyttäjä ymmärtää niiden vastaavan samanlaista toiminnallisuutta. Tätä samankaltaisuutta vahvistaa myös se, että esimerkiksi edellä mainittuja plus-ikoneja klikkaamalla tulee esiin varsin samanhenkiset kaavakkeet ruudun yläreunaan. (Parkkinen 2002, 43–44.)

Tallentaminen itsessään on toteutettu perinteisin kaavakkein. Näytön lisäämisessä annetaan nimi ja näytölle kansio, mediasisällössä valitaan tiedosto käyttäjän koneelta tai kirjoitetaan WWW-osoite. Annetut tiedot lähetetään PHP-skriptille, joka tarkistaa,

että tiedot ovat järjestelmän hyväksymässä muodossa (esimerkiksi tiedostotyyppi ei poikkea sallituista) ja tallentaa lähetetyt tiedot oikeisiin paikkoihin.

4.6 Muokkaaminen

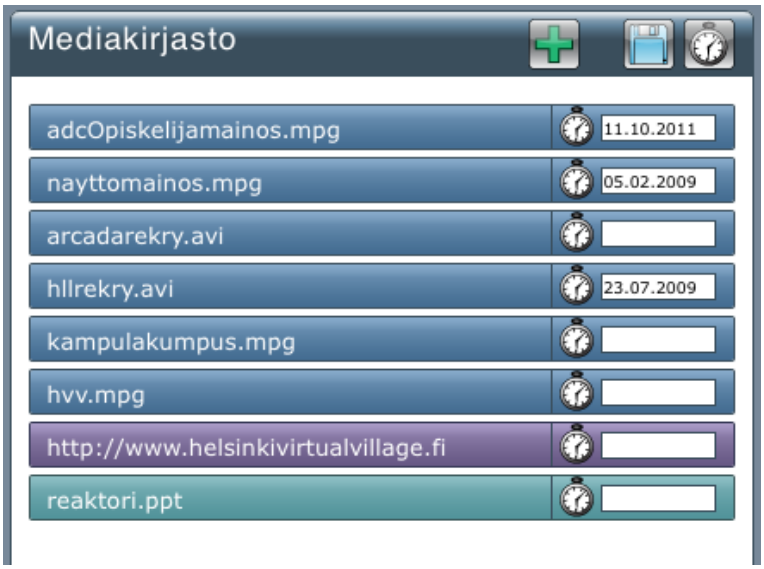
Muokkaamisella Turbanissa tarkoitetaan näyttöjen tietojen muuttamista ja mediasisällön poistoaikojen muuttamista. Näyttöjen tiedoista voi muuttaa nimeä ja kansiota, jos esimerkiksi näyttöä luodessa tuli tehtyä kirjoitusvirhe tai näytön tiedot muuttuvat muusta syystä, esimerkiksi sama näyttö siirretään eri tiloihin, mutta näyttöillä näkyvä materiaali eli soittolista ei muutu. Näyttöjen muokkaamiseen paradigmaksi on otettu kynä-ikoni. Sitä klikkaamalla siirrytään muokkaustilaan ja hiiren kursori muuttuu kynän näköiseksi, kun sen vie näyttöjen kohdalle.

Mediasisältö on monesti sellaista, että sitä olisi tarve näyttää vain tiettyyn päivämäärään asti. Esimerkiksi teatteriesityksen mainos, jossa kerrotaan, että esitys on välillä 1.5.2009–1.6.2009. Kuten luvussa 2 mainittiin, Urbaanimedian ylläpitäjät eivät usein muistaneet poistaa vanhentunutta mediasisältöä näyttöjen soittolistoilta. Siispä rakensin Turbaniin toiminnon, jolla kullekin mediasisällölle voidaan määrittää aika, jonka jälkeen se poistetaan kaikilta soittolistoilta. Tämä tapahtuu klikkaamalla mediakirjaston yläpalkin kello-ikonia, jolloin mediasisällön reunaan tulee tekstikenttä, johon voi syöttää päivämäärän. Poistoajat voi tallentaa klikkaamalla kello-ikonin viereen ilmestynyttä levyke-ikonia.

Poistoaikojen muokkaamisessa hyödynnetään hahmolakeja eli ihmisen havaintojärjestelmän tapoja ryhmitellä ärsykkeitä isommiksi kokonaisuuksiksi. Läheisyyden lain mukaan kaksi toistensa lähellä olevaa asiaa mielletään yhteen kuuluvaksi. Tästä syystä poistoaikojen muokkaamistilaan siirryttäessä tulee mediasisällön lisäämisnapin ja poistoaikojen muokkaamiseen tarkoitettujen tallenna- ja muokkaaminen päälle/pois -nappien (kuva 10) väliin selvä väli, jolloin lisäämisnappi erotetaan selvästi omaksi kokonaisuudekseen (kuva 11). Vaikka kunkin mediasisällön reunaan tuleva kelloikoni (kuva 12) tekee käyttöliittymän hieman sotkuisemman näköiseksi, kelloikonit ovat kuitenkin oleellisia, sillä näin niiden vieressä oleva tekstikenttä mielletään poistoaikoihin liittyväksi. Samoin mediasisällön reunaan tuleva tekstikenttä ymmärretään mediasisältöön liittyväksi. (Sinkkonen, Kuoppala, Parkkinen, Vastamäki 2006, 89–92.)

Kuva 10. Mediakirjaston ikonit ennen poistoaikojen muokkaamisnapin painamista.

Kuva 11. Mediakirjaston ikonit poistoaikojen muokkaamisnapin painamisen jälkeen.



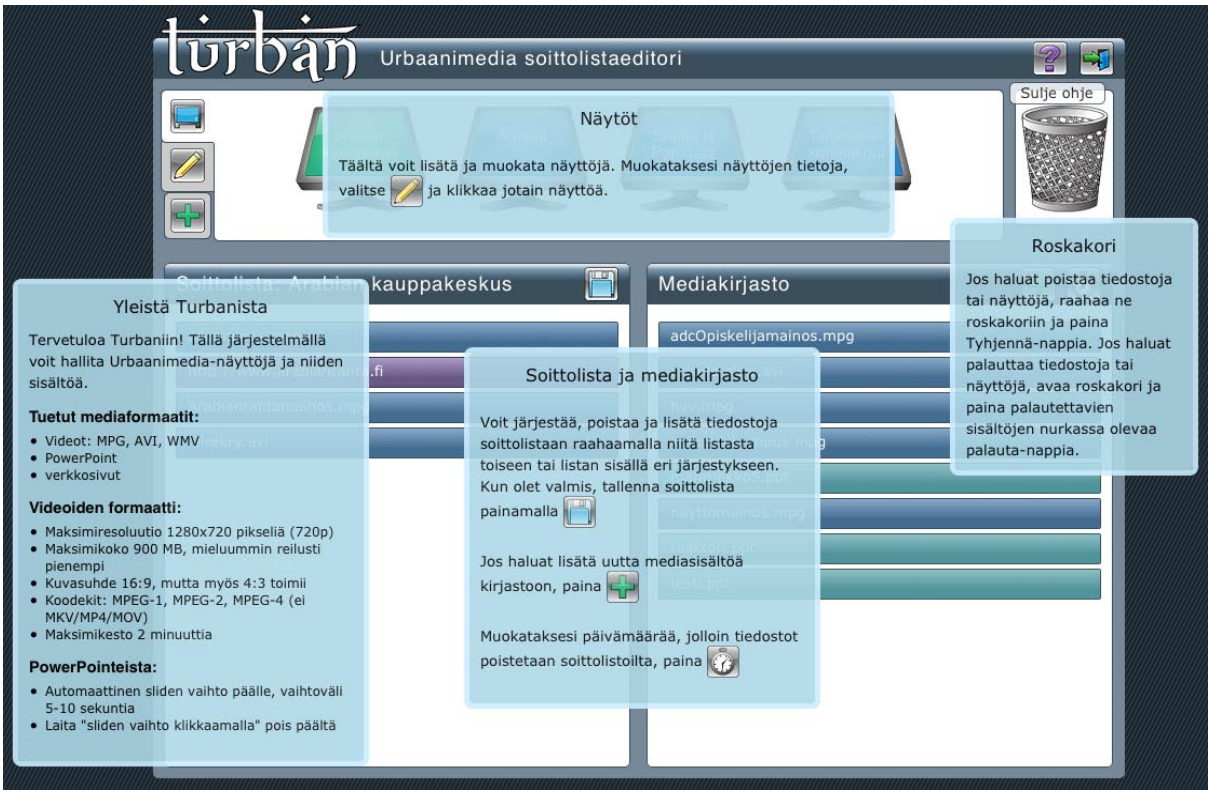
Kuva 12. Mediasisällön poistoaikojen tekstikentät.

4.7 Opasteet

Koska ikonien tulkinta voi vaihdella ja kaikki toiminnot eivät pelkästään silmäilemällä havainnollistu käyttäjille, päätin sisällyttää Turbaniin aputoimintoja, joilla kerrotaan toimintojen funktiot selkeästi, mutta niin, ettei käyttäjä häiriinny. Ensimmäinen aputoiminto on rakentamani jQuery Tooltip -plugin, jolla halutuille elementeille saadaan näytettyä vihjeteksti hiiren kursorin ollessa elementin päällä. Kuvassa 13 näkyvässä tooltip-opasteessa vihjeteksti näkyy läpinäkyvänä laatikkona hiirikursorin alla. Turbanissa tätä käytetään kaikkien ikonien opasteena. Syy, miksi valitsin jQuery pluginin toteutustavaksi oli, että samaa Tooltip-toimintoa voisi hyödyntää muissa ohjelmissa ja lisäksi halusin opetella, miten jQueryn toiminnallisuuden kasvattaminen pluginien avulla tapahtuu. Ainoa haittapuoli Tooltip-toiminnossa on, että se ei toimi kosketusnäytöillä eikä pelkästään näppäimistöä käytettäessä.

Kuva 13. Napin tooltip-opaste.

Lisäksi koin, että Turbaniin liittyvät olennaiset tiedot, kuten tuetut tiedostomuodot, drag&dropin toiminta yms. olisi syytä löytyä yhdestä paikasta. Tätä varten rakensin Turbaniin toiminnon, joka näyttää, mitä kukin Turbanin käyttöliittymän osio tekee ja havainnollistaa tärkeimpien ikonien toiminnan (kuva 14). Tavoitteena oli helpottaa Urbaanimedian käytön koulutusta ja myöskin auttaa Turbanin käyttäjiä löytämään esimerkiksi tuetut tiedostoformaatit ja kuvakoot, jos näitä tietoja tarvitsee lähettää niille tahoille, jotka luovat sisältöä Urbaanimedianäytöille.



Kuva 14. Turbanin ohjeet.

5 TOTEUTUS

Lähdin rakentamaan Turbania 37 Signalsin Jason Friedin “Signal vs. Noise” blogissa esittämällä Epicenter Design mallilla. Tässä mallissa määritetään ensin esimerkiksi verkkosivuston keskeisin asia ja lähdetään rakentamaan verkkosivustoa tästä keskustasta ulospäin. Turbanissa tämä keskeisin asia oli soittolistojen päivittäminen, joten aivan ensimmäiseksi rakensin soittolistan ja mediakirjaston ja näiden välille drag-and-drop-toiminnallisuuden soittolistojen päivittämiseen. Tästä pohjasta lähdin kasvattamaan Turbania ominaisuus kerrallaan, luomalla kunkin näytön soittolistan lataamisen, tallennusominaisuudet ja lopulta muokkaamisominaisuudet sekä lopullisen ulkoasun. (Fried 2004.)

Turbanin rakentamiseen tekniikoiksi valikoitui Javascript, HTML, CSS ja PHP, sillä halusin järjestelmän, jota voisi tarvittaessa käyttää esimerkiksi kannettavalla tai hyvällä WWW-selaimella varustetulla puhelimella. Esimerkiksi jonkin Urbaanimedianäytön huollon yhteydessä Turbania voitaisiin käyttää näytön toiminnan testaamisessa apuna näytön sijoituspaikassa. Tähän verkkosovellus on ehdottomasti paras vaihtoehto, sillä näin tietokoneelle ei tarvitse asentaa minkäänlaista ohjelmaa, kunhan WWW-selaimen tuki HTML:lle, Javascriptille ja CSS2/3:lle on riittävän hyvä.

Tekniikka rajoitti hieman toteutusta. Aluksi sovellus oli tarkoitus asentaa vanhalle mediapalvelimelle, jolla ei kuitenkaan ollut mahdollista saada esimerkiksi MySQL-tietokantaa käyttöön, joten toteutin Turbanin tiedon tallennuksen käyttäen tekstitiedostoja. Myöhemmässä vaiheessa ADC:lla päätettiin, että mediapalvelin olisi kannattavaa päivittää parempaan. Tämän jälkeen olisi ollut mahdollista käyttää niin MySQL-tietokantoja kuin myös esimerkiksi Ruby On Railsia. Tässä vaiheessa Turban oli kuitenkin jo tehty pitkälle PHP:lla ja tekstitiedostoilla, joten en nähnyt suurta hyötyä koko projektin kääntämisessä Ruby On Rails -alustalle tai tekstitietokannan siirtämisestä MySQL-tietokantaan tässä projektin vaiheessa.

5.1 Selaintuki

Selaintuen päätin rajata heti aluksi vain nykyaikaisiin selaimiin. Koska Art and Design City Helsinki Oy:n käytössä on tätä kirjoitettaessa pääasiassa Firefox 3.x.x, valitsin sen pääkehitysalustaksi jo siihen saatavien debug-työkalujen (esimerkiksi Firebug) monipuolisuuden takia. Projektin valmistuessa sain mukaan myös tuen Google Chromelle ja Safarille (todennäköisesti myös muille Webkit-renderöintimoottoriin

perustuville selaimille) sekä Operalle ja Internet Explorer 8:lle. Internet Explorer 6 ja 7 rankattiin pois niiden puutteellisen CSS-tuen ja hitaan Javascript-moottorin takia, kuin myös siksi, että yhteensopivuuden rakentaminen olisi vaatinut kohtuuttomasti työtä. Turbanin käyttäjät eivät kuitenkaan kyseisiä selaimia käytä. Applen iPhonella ei – kuten ei todennäköisesti muillakaan kosketusnäyttöpuhelimilla – Turbanin käyttö onnistu, sillä puhelimet eivät tue drag-and-dropia, koska kyseistä kosketuselettä käytetään ruudun vieritykseen.

5.2 AJAX

AJAX, eli Asyncronous JavaScript and XML, on joukko tekniikoita, joilla voidaan lähettää ja vastaanottaa tietoa WWW-selaimen ja palvelimen välillä ilman sivujen latauksia. Tämä tapahtuu pääasiassa hyödyntämällä XMLHttpRequest- tai IFrame-objekteja. Turbanissa käytetään molempia objekteja eri yhteyksissä jQuery-kirjaston tarjoamien AJAX-kutsujen käsittelyn helpottamiseen tarkoitettujen funktioiden avulla. AJAXin avulla lähetettävän datan muotona Turbanin AJAX-kutsut käyttävät enimmäkseen JavaScript Object Notation (JSON) -muotoa, mutta tietyissä tapauksissa myös pelkkää tavallista tekstimuotoa. (Wikipedia 2009, AJAX.)

AJAXin käytön perustelen sillä, että halusin Turbanista verkkosovelluksen, joka toimii tasan yhdessä ruutunäkymässä ilman sivun vaihtoja. Näin koko Turbanin käyttöliittymä pysyy selkeämpänä, sillä sivuston layout ei muutu yhtäkkiä ja käyttäjä ei pysty harhailemaan sivustorakenteessa niin, ettei löydä enää takaisin alkuun. Kirjautuminen on tästä ainoa poikkeus, sillä se toimii eräänlaisena porttina Turbaniin ja sen toteuttaminen on turvallisempaa pelkän PHP-skriptauksen avulla. Kirjautumista lukuun ottamatta kaikki PHP-skriptin käyttöä vaativat toiminnot suoritetaan taustalla tapahtuvien AJAX-kutsujen kautta. Käytännössä tämä tarkoittaa seuraavia toimintoja:

- Mediasisällön sekä näyttöjen tallentaminen ja poistaminen.
- Soittolistojen lukeminen ja kirjoittaminen.
- Mediakirjaston tietokannan lukeminen, kirjoittaminen ja muokkaaminen.
- Mediasisällön poistoaikojen muokkaaminen.
- Toimintojen ilmoitusten ja virheilmoitusten välittäminen selaimeen.

Koska AJAXia käytettäessä käyttäjälle ei vakiona anneta minkäänlaista ilmoitusta AJAX-kutsun suorittamisen tilasta, on tärkeää käyttää latausindikaattoreita kertomaan tästä.

Turbanissa tähän käytetään latausanimaatiota, jota näytetään kun odotetaan viestijärjestelmässä AJAX-kutsujen suorituksen päättymistä. AJAXilla kutsuttu PHP-skripti määrittelee kunkin Turbanin toiminnon onnistumisen ja epäonnistumisen ja ilmoittaa siitä paluuviestinä AJAXin avulla, joko antamalla käyttäjälle kiitoksen, lisäinfoa tai sitten virheilmoituksen toiminnon epäonnistuessa.

5.3 jQuery

jQuery JavaScript-kirjasto ei ollut minulle tuttu projektin alussa. Olin aiemmin käyttänyt muissa projekteissa Prototype-kirjastoa, joka integroituu näppärästi muun muassa Ruby On Railsiin. Koska Railsia ei ollut mahdollista käyttää projektin alkaessa, päätin tutkia muitakin vaihtoehtoja. Tuolloin alan blogeissa ym. puhuttiin paljon John Resigin kehittämästä jQuerysta, sen helppokäyttöisyydestä ja monipuolisuudesta. Lisäksi kirjastoon pohjautuvalle JavaScript-koodille löytyi jo tuolloin melko paljon dokumentaatiota ja koodiesimerkkejä.

Lisähyötynä oli jQueryn kehittäjien kanssa yhteistyössä luotu jQuery UI efekti- ja interaktio-kirjasto. Prototypea laajentava vastaava kirjasto oli script.aculo.us, mutta tuolloin siitä ei löytynyt aivan samalla tavalla toiminnallisuutta kuin jQuery UI:ssa ja sen syntaksi oli hieman monimutkaisempi. Tiesin, että tulisin hyödyntämään Turbanin käyttöliittymässä aiemmin script.aculo.us-kirjaston kautta minulle tutuksi tulleita liu'utus-, häivytys- ym. efekteja ja ennen kaikkea interaktio-toiminnallisuutta, kuten drag&drop, järjesteltävät elementit ym. jQuery UI tarjosi nämä ja paljon muuta, joten jQueryn ja jQuery UI:n yhdistelmä tuntui kokonaisuutena parhaalta ratkaisulta ja sellaiseksi se osoittautuikin ajan myötä.

5.4 Tietokanta

Koska MySQL-tietokantaa ei ollut tarjolla, päätin tallentaa Turbanin käyttämät tiedot tekstitiedostoihin. Tästä on toisaalta myös se hyöty, että toisin kuin tietokantainjektiohyökkäyksillä, tekstitiedostosta luettaessa ei ole mahdollisuutta päästä syöttämään tietokantaan esimerkiksi käyttäjätiedot paljastavaa tai muokkaavaa tietokannan käsittelykoodia. Lisäksi Turbanin tietokannan datan määrä on sen verran pieni, että MySQL ei tarjoaisi juuri nopeusetujakaan tekstitiedostojen lukemiseen, jäsentelyyn ja kirjoittamiseen verrattuna. Myös tekstitiedostopohjaisen tietokannan hallinta käsin on helppoa, sillä muokkaaminen onnistuu millä tahansa tekstieditorilla ja data on helposti ihmisen luettavassa muodossa.

Haittapuolia on taas lähinnä tietokannan muokattavuudessa. Virhetilanteessa koko tietokantatiedosto saattaa pahimmillaan tyhjentyä tai sitten korruptoitua niin, ettei sen muokkaaminen onnistu kuin käsin tekstieditorilla. Tästä syystä kiinnitin paljon huomiota juuri tallennuksen, muokkaamisen ja poiston testaukseen erilaisia virhetilanteita simuloiden. MySQL:lla esimerkiksi jonkin tietyn tietokantaelementin muokkaus on helpompaa, eikä koko tietokantaa tarvitse lukea muistiin ja kirjoittaa joka kerta uusiksi sen jälkeen, kun siihen on tehty muutoksia.

Käytännössä Turbanin tekstitietokantoihin tallennetaan lähes pelkästään kuvailutietoa. Itse mediasisältö (videot, powerpointit) on verkko-osoitteita lukuun ottamatta edelleen omissa tiedostoissaan mediakirjaston hakemistossa. Kuvailutieto on tietoa sisällöstä, tässä tapauksessa tiedostonimiä, näyttöjen nimiä tai verkko-osoitteita, tiedostojen tyyppejä tai hakemistojen nimiä ja tiedostojen poistoaikoja. Näitä tietoja käytetään kaikissa Turbanin sisältöelementeissä. (Samela 2002, 62-75.)

Turbanin tietokantojen muoto on sellainen, että yhdelle riville kirjoitetaan yhteen elementtiin (esimerkiksi Urbaanimedianäyttö) liittyvät tiedot, seuraavalle seuraavan elementin kaikki tiedot jne. Tiedot jaetaan ||-merkillä, joka antaa kullekin tiedolle oman lokeronsa ja määrittää myös erilaisten tietojen järjestyksen. Kun PHP-skripti lukee tekstitietokannan, se jakaa kunkin rivin niin, että palkkien välissä olevat tiedot tallennetaan omiin nimettyihin muuttujiinsa ja näin luetuista tiedoista muodostuu taulukkorakenne, jota voidaan käsitellä eri tavoin ja sitten lähettää takaisin selaimeen JavaScriptin käsiteltäväksi.

Kuvassa 15 on esimerkki tietokannan rakenteesta. Esitetty rakenne kuvaa mediakirjastoa, jossa on neljä mediasisältöä. Rakenne on “tiedostonimi || tiedoston tyyppi || tiedoston poistoaika”. Jos poistoaikaa ei ole asetettu (tiedostoa ei poisteta soittolistoilta koskaan), se merkitään 00.00.0000.

```
videoklippi.mpg||video||24.12.2010
videoklippi2.wmv||video||00.00.0000
powerpointABC.ppt||powerpoint||00.00.0000
http://www.metropolia.fi||url||05.09.2010
```

Kuva 15. Tietokannan rakenne.

5.5 Tietoturva

Sinänsä pahin, mitä Turbanilla saa aikaan väärissä käsissä, on jonkin sopimattoman videon saattaminen näytöille pyörimään. Tästä ei sinänsä ole suurta huolta, sillä palvelun kaikki tiedot ovat visusti ADC:n hallussa ja pääsy palveluun yleensäkin rajattu. Pääsy Turbaniin kuin myös mediapalvelimelle on käyttäjätunnuksen ja salasanan takana. Tekstitiedostoihin pohjautuvan tietokannan helposti luettava muoto luo tietoturvariskin, mutta toisaalta Turbanin tietokannoissa ei kuitenkaan ole mitään kriittistä tietoa.

Muuten Turban rajoittaa, mitä tiedostoja voi tallentaa tarkistamalla niiden MIME-tyypit ja estää myöskin kaiken JavaScript- tai PHP-koodin syöttämisen esimerkiksi verkko-osoitteiden tallentamisen yhteydessä. JavaScript-koodi on myös pakattu, jotta se ei ole helposti ihmisten luettavassa muodossa.

6 ULKOASUN KEHITYSKAARI

Kun olin ratkaissut, miten Turbanin käyttöliittymä rakentuu, aloin kehittää ulkoasua eteenpäin. Aivan ensimmäinen idea oli ADC:n logon ja Arabianranta.fi-portaalin värimaailmaa kopioiva tiilenpunaisen, harmaan ja valkoisen yhdistelmä. Päädyin kuitenkin lopulta toisiin ratkaisuihin, sillä Turbania ei ollut oleellista brändätä mitenkään Arabianrantaan tai ADC:hen liittyväksi.

Monet verkkopalvelut (esim. Twitter) hyödyntävät miellyttäviä pastellisävyjä, joten kokeilin rakentaa Turbanin niillä, sisällyttäen paljon tyhjää tilaa, pyöristettyjä kulmia ja litteitä pintoja. Totesin lopulta, että tämä oli liian valjun näköinen ratkaisu Turbaniin, pastellisävyistä ei saanut riittävän suuria sävyeroja esimerkiksi erilaisten mediatyyppien merkeiksi. Kakkosversiosta jäi jäljelle lähinnä suuret fontit, pyöreät kulmat ja tyhjän tilan käyttö.

Kolmas ja viimeinen versio pohjasi viileisiin, skarppeihin sinisen sävyihin ja hillittyihin gradientteihin. Sinkkosen mukaan viileät värit vetäytyvät taustalle, kun taas kirkkaat värit sopivat pieninä pintoina korostamaan asioita. Tästä syystä Turbanin käyttöliittymän taustalla on viileää sinistä, kun taas ikoneissa on voimakkaampia, kirkkaampia värejä, jotta huomio kiinnittyisi niihin. Sinkkonen kuitenkin kertoo, että vahvoja värejä ei olisi syytä käyttää usein käytettävissä käyttöliittymissä. Mielestäni Turbanissa tämä toteutuu varsin hyvin, sillä värimaailma ei ole toisaalta tylsä muttei

mitenkään räikeäkään. (Sinkkonen ym. 2006, 126–133.) Lopullinen ulkonäkö on riittävän lähellä työpöytäsovellusten ulkonäköä, mutta erottuu kuitenkin selkeästi verkkosovellukseksi, sillä täysin työpöytäsovelluksen näköinen verkkosovellus saattaisi hämmentää käyttäjää, jos sovellus ei yhtäkkiä toimikaan esimerkiksi katkenneen verkkoyhteyden takia. Myöskään käyttöjärjestelmien käyttöliittymäelementtien erilaiset ulkonäöt eivät näin sotke Turbania esimerkiksi Mac- tai Windows-työpöytäohjelmaksi.

Grafiikka syntyi pääasiassa Adobe Illustrator CS3:lla. En halunnut käyttää esimerkiksi valmiita ikonikirjastoja, sillä halusin yhtenäisen ulkoasun, joka olisi tehty loppuun asti omilla ehdoillani, eikä esimerkiksi ikonikirjaston muotokieltä mukaillen. Mediakirjaston ja soittolistan sisältöjä kuvaavissa palkeissa hyödynsin niiden skaalautuvuuden (esimerkiksi pitkä tiedostonimi tai verkko-osoite) varmistamiseksi uudemmista selaimista löytyvää tukea CSS-liukuväreille.

6.1 Napit

Kokeilin aluksi nappeja, joissa on sekä ikoni että seliteteksti samassa. Totesin kuitenkin, että nämä vievät turhan paljon tilaa ruudulla, joten päädyin pelkästään ikoneita hyödyntäviin nappeihin kaikissa muissa paitsi roskakorin tyhjennä-napissa ja viestijärjestelmässä näkyvissä napeissa. Näissä napeissa käytin vain selitetekstiä, sillä ne sopivat kontekstissaan paremmin käyttöliittymään (leveä, suuri ja selaimista jo tuttu nappi vs. kuution muotoinen pieni ikoninappi) selvästi muista napeista poikkeavalla ilmeellään, ovathan näiden nappien toiminnotkin kuitenkin peruuttamattomampia kuin muiden nappien.

Kun pelkästään ikonin sisältävä nappiformaatti oli valittu, kokeilin ensin mallia, jossa ikonit olivat kolmiulotteisina. Tämä toimi hyvin niin kauan kun napit olivat riittävän isoja, mutta haluamassani koossa (30x30 pikseliä) niistä tuli epäselviä, eikä kolmiulotteisuus visuaalisena tehokeinona tarjonnut minkäänlaista etua käytettävyyden suhteen.

Ikonit saivat lopullisen muotonsa Applen iPhonen innoittamana. Sen iOS-käyttöjärjestelmässä käytetään ikoneita, joiden koko ja muoto on kutakuinkin sormenpäälle ideaalinen. Vaikka Turbania ei ole suunniteltu mitenkään kosketusnäytöt huomioivaksi, iPhone-henkiset napit sopivat siihen hyvin, sillä ne vievät verrattain vähän tilaa ja ovat pääasiassa helposti ymmärrettäviä, kunhan ikonien symboliikka on riittävän selkeää ja yleismaailmallista.

Ikonien muotokieli on pitkälti muista ohjelmista tuttua. Diskettiä on jo vuosikaudet käytetty kuvaamaan tallentamista, vaikka diskettiä ei tallennusmuotona enää juuri käytetäkään. Kynä taas symboloi muokkaamista, plus-merkki lisäämistä. Kysymysmerkkiä on monesti käytetty esimerkiksi ohjelman ohjetiedostojen näyttämiseen. Kello-ikoni symboloi aikaa, siispä se on looginen vastine poistoaikojen muokkaamiselle. Välilehdissä käytetty näytön valintaa symboloiva näyttö ei ehkä ole paras mahdollinen metafora ja uloskirjautumista kuvaava ovesta sisään menevä nuolikin on hieman kyseenalainen, mutta kuitenkin monesti käytetty symboli. Garrettin (2003) mukaan käyttöliittymissä ei kannattaisi hyödyntää liikaa tosielämän objekteihin liittyviä metaforia, sillä kulttuurieroista johtuen ne voidaan ymmärtää väärin.

Tein ikonit Adobe Illustrator CS3:lla vektorigrafiikkana, sillä vektorigrafiikkaa voi suurentaa ja pienentää vapaasti ilman, että grafiikan tarkkuus huononee. Tämä ei kuitenkaan tarkoita sitä, että samaa ikonia pystyisi käyttämään missä tahansa koossa. Ikonit tulisi suunnitella sovelluksessa käytettyihin kokoihin, sillä liian yksityiskohtainen ikoni näyttää sekavalta pienessä koossa, kun taas yksinkertaisempi ikoni näyttää liian pelkistetyltä suurennettuna. Hyvin pienikokoiset ikonit vaativat usein poikkeuksellisen pelkistettyä muotokieltä, jotta hyvin rajoittunut pikselimäärä (esimerkiksi Windowsin pienet 16x16 pikselin ikonit) pystytään käyttämään niin, että ikonin merkityksen ymmärtää. Tämän voi toteuttaa esimerkiksi rajoittamalla perspektiiviä ja ikonin elementtejä. (Hodge 2008.)

Kun ikonit olivat valmiit, niistä tehtiin vielä erikoisversiot vaihtamalla napin pohjan liukuväri toisinpäin. Näitä versioita käytetään ns. hover-tilassa, eli silloin kun hiiren kursori on ikonin päällä. Näin napin affordanssina nähdään mahdollisuus painaa sitä. Lopuksi ikonit skaalattiin 30x30 pikselin kokoon ja tallennettiin bittikarttamuodossa. (Ks. kuva 16.) Illustratorin bittikarttatallennuksen laatu oli kuitenkin huono; ikonien yksityiskohdat sumentuivat, muodot muuttuivat ja väriliukuihin tuli selviä rajoja. Ratkaisin ongelman kierrättämällä ikonit suurikokoisina Adobe Photoshop CS3:n kautta ja pienentämällä ne sen avulla ennen tallennusta. Ilmeisesti ainakaan Illustrator CS3:ssa vektorien muuntaminen bittikartoiksi ei toimi oikein pienillä pikselimäärillä.

Kuva 16. Turbanin ikoneita. Alemmassa rivissä kunkin ikonin hover-tila.

6.2 Välilehdet

Välilehdet ovat erinomainen valinta moniin käyttöliittymiin, sillä niissä on suora yhteys fyysisen maailman käyttötapoihin ja niiden avulla on helppo jakaa tietoa erilaisiin osioihin. Välilehdet ovat myös tuttu paradigma nykyaikaisista selaimista ja monilta verkkosivuilta, joten niiden toiminta verkkosovelluksessa vaatii käyttäjältä hyvin vähän pohdintaa. Välilehtien kanssa esitystapa on kuitenkin hyvin tärkeä, kuten seuraavasta selviää.

Päätin kokeilla näyttöjen hallintaosiossa vertikaalisia välilehtiä, joilla voidaan valita joko soittolistan valintamoodi tai näyttöjen editointimoodi. Välilehdet yhdistävät edellä mainitut toiminnot yhdeksi kokonaisuudeksi, joten ne mielletään samaan asiaan vaikuttaviksi. Sen sijaan näyttöjen lisäämiseen tarkoitettu nappi eriytettiin, sillä se ei sopinut lainkaan välilehtijärjestelmään, koska se käyttää kuitenkin viestijärjestelmää näyttöjen lisäämiskaavakkeisiin, kuten muutkin lisäämistoiminnot. (Ks. kuva 17.)

Ongelma tässä järjestelyssä oli, että valintamoodin ja editointimoodin välille ei loppujen lopuksi synny tarpeeksi eroa, sillä välilehtien vaihtuessa näyttö-osion sisältö ei kuitenkaan vaihdu, kuten yleensä välilehtiä käytettäessä, vaan pelkästään kursorin ulkonäköä käytetään viestittämään valitusta moodista: tavallinen nuoli- tai sormikursori valintamoodissa, kynäkursori editointimoodissa. Valitettavasti vasta ominaisuuden lisäämisen jälkeen huomasin myös, että erilliset kursorit toimivat vain Windows-käyttöjärjestelmillä. Mac OSX:ssa ne eivät toimi tällä hetkellä millään selaimella, joten toimintoja selkeyttävät kursorit jäivät valitettavasti osittain toimimattomiksi.

Kuva 17. Näyttöosion välilehdet.

Samalla huomasin, että tässä ominaisuudessa yhdistyi kaksi erilaista paradigmaa: välilehdet ja esimerkiksi kuvankäsittelyohjelmista tuttu työkalurivi, jossa kaikki käytettävissä olevat työkalut ovat näkyvillä. On valintatyökalu sekä muokkaustyökalu ja lisäksi vielä työkalu lisäämiseen. Vaikka tämä tavallaan toimii ja lisäysnapin eristäminen välilehdistä auttaa myös, on toiminnossa selvästi parantamisen varaa.

Sen sijaan välilehdet sopivat hyvin sisällön lisäämiseen kirjastoon. HTML:n teknisestä rajoitteesta johtuen tarvittiin erillinen kenttä tiedostojen lisäämiselle ja WWW-osoitteiden lisäämiselle. Jotta nämä saatiin mahdutettua siististi viestijärjestelmän viestin tilaan, olivat välilehdet hyvä ratkaisu. Ensinnäkin toteutettiin edellä mainittu yhteenkuuluvuus, mutta toisaalta tiedoston ja WWW-osoitteen lisäämiselle saatiin selvä ero välilehtien hieman erilaisten sisältöjen avulla. Kuvassa 18 näkyvät välilehdet tyylittelin siten, että valittu välilehti näyttää kuuluvan osaksi sen alla olevaa kaavakeosiota, kun taas valitsematon välilehti nähdään tummemman taustavärinsä takia olevan taustalla, esillä olevan välilehden takana. (Krug 2006, 79–84.)

Kuva 18. Sisällön lisäämisruudun välilehdet.

7 MITEN TURBANIA VOISI KEHITTÄÄ ETEENPÄIN?

Turbaniin olisi syytä saada jonkinlainen pääkäyttäjän (administrator) työkalu, jolla voisi muun muassa muokata näyttö-, kirjasto- ja soittolistatiedostoja tekstimuodossa, mikäli järjestelmässä tapahtuu jokin pahempi virhe. Näin voisi varmistaa, että kriittisen virheen sattuessa ei tarvitse ottaa yhteyttä minuun, vaan käyttäjät voivat yrittää selvittää sen itsekin. Vaikka Turban onkin rakennettu niin, ettei sen teoriassa pitäisi hajota helposti, on aina mahdollista, että jokin asia aiheuttaa peruuttamattoman virheen, jota ei saa korjattua vain tallentamalla vaikkapa soittolista uudelleen.

Tiedosto-operaatioita yms hallitsevasta PHP-koodista olisi syytä saada selkeämpää ja siistimpää. Luokkien käyttö ei tässä yhteydessä tarjoa oikeastaan muuta kuin vähemmän koodin toistoa. Tätä voisi parantaa tekemällä luokkien instanssien luomisesta sellaisen, että instansseille tarvitsisi syöttää selvästi vähemmän muuttujia. Näin myös backendista saataisiin modulaarisempi. Tämän opinnäytetyön tekemisen aikana JavaScript-puoli päivitettiinkin jo useampaan otteeseen uudempien jQuery- ja jQuery UI -kirjastojen julkaisujen tahdissa. Tämä paransi Turbanin nopeutta hieman, joten jatkossa vastaava päivittäminen tarvittavine koodimuutoksineen olisi järkevää.

Turbanin modulaarisuutta voisi kehittää myöskin eteenpäin, jolloin voisi esimerkiksi määritellä käytetäänkö roskakori-komponenttia vai ei, miten monta soittolistakomponenttia on käytössä jne. Näin Turbanin koodia voisi taivuttaa muihinkin käyttötarkoituksiin, esim. verkkopohjaiseen, iTunes-henkiseen musiikkisoittimeen. Komponenteista voisi myös tehdä yleisluontoisia jQuery-plugineita, jolloin niitä voisi käyttää yksittäin tai yhdessä.

Viestijärjestelmää voisi tehdä huomaamattomammaksi. Toisaalta viestien pitää näkyä, mutta niiden ei tarvitsisi kuitenkaan tulla aina muiden elementtien eteen. Esimerkiksi Gmail-tyyppiset yhden rivin viestit tekisivät muun muassa soittolistan tallennuksen viesteistä vähemmän häiritseviä.

Myös tukea moderneille kosketusnäyttöpuhelimille voisi kehittää. Kuten Toteutus-osiossa mainittiin, kosketusnäyttöpuhelimet eivät tue drag-and-drop-toiminnallisuutta mikä johtuu niiden tavasta käyttää vastaavaa kosketuselettä ruudun vierittämiseen pysty- ja vaakasuunnassa. Muuten Turban toimii ainakin Applen iPhonelle aivan kelvollisesti. Drag-and-drop-ongelman voisi korjata muuttamalla sitä hyödyntävän toiminnallisuuden siten, että kun Turban havaitsee, että käytössä on

kosketusnäyttöpuhelin, se muuttaisi drag-and-drop-toiminnallisuuden esimerkiksi Brian Swartzfagerin blogissaan esittämälle tavalle, jossa klikkauksilla voisi siirrellä mediasisältöä. Swartzfagerin menetelmällä yksi klikkaus valitsee siirrettävän objektin. Tämän jälkeen valitaan kohde, johon objekti siirretään toisella klikkauksella. (Swartzfager 2008)

Tiedostojen tallentaminen tulisi myös tehdä sellaiseksi, että se tapahtuu taustalla. Suuria usean sadan megatavun tiedostoja tallennettaessa selain käytännössä jumittuu tiedonsiirron ajaksi, jolloin käyttäjä saattaa luulla, että selain on jumittunut.

8 YHTEENVETO

Turban-projektin myötä opin huomattavasti lisää JavaScript-ohjelmoinnista ja erityisesti jQuerysta tuli tuttu ja rakas kirjasto, jota hyödynsin muissakin projekteissa opinnäytetyöni ohella ja sitä alettiin käyttää myös ADC:n Arabianranta.fi-portaalissa tiettyihin toimintoihin. Turbanin kehittäminen myös opetti paljon verkkosovelluksen käytettävyyden suunnittelusta ja haasteista. Sain paremman ymmärryksen siitä, miten jokin yksinkertaiselta vaikuttava ominaisuus voi helposti paisua vaikeaksi toteutettavaksi, koska se nivoutuu niin moniin muihin ominaisuuksiin.

Haasteellisinta oli varmasti aavistaa kaikki mahdolliset kombinaatiot, joita käyttäjä saattaisi tehdä. Tämän olisi voinut ratkaista tekemällä Turbanista käyttäjää kädestä pitävän “velho”-tyyppisen ratkaisun, jossa valitaan tehtävä ja vaihe vaiheelta viedään käyttäjä tehtävän toimintaketjujen lävitse, mutta usein Urbaanimedianäyttöjä päivittäessä on tarve tarkistaa muitakin asioita samalla kertaa. Lisäksi kun käyttäjä on useamman kerran viety dialogiryppään läpi, haluaisi hän todennäköisesti ennemminkin käyttää jotain kaikki optiot heti tarjoavaa ratkaisua. Tämä on todettu ADC:n käyttämän ryhmäsähköpostiohjelman käytössä: joka kerta kun jokin ryhmäsähköposti täytyy lähettää, marisevat käyttäjät samojen tietojen täyttämisestä yms.

Yksi haastavimmista tehtävistä oli tiedostojen hallinnan saaminen riittävän luotettavaksi, ettei järjestelmä esimerkiksi vahingossa poista näyttöä, koko soittolistaa tai vaikka kaikkia videotiedostoja. Garrettin (2003, 92-93) mukaan virhetilanteita varten on hyvä varautua selkein virheilmoituksin ja varoituksin. Järjestelmän piti siis pystyä reagoimaan virhetilanteisiin ja välittämään siitä myös käyttäjälle järkevää palautetta, eikä pelkkiä kehittäjälle aukenevia virheilmoituksia. Ideaalista toki olisi, että

virheilmoituksia ei tulisi kuin korkeintaan väärän tiedostotyypin tallennuksessa, mutta varsinkin suuria tiedostoja tallentaessa käyttäjälle ei voida helposti tarjota tietoa tallennuksen etenemisestä vaan se jää selaimen indikaattorien harteille. Tämän opinnäytetyön yhteydessä ei myöskään ehditty rakentaa Turbaniin minkäänlaisia varmuuskopiotoimintoja, joista voitaisiin virheen sattuessa palauttaa tietoja. Toisaalta soittolistojen tallentaminen uudelleen ei vaadi kovin montaa minuuttia, mutta mediakirjaston luominen sen sijaan voi kestää, jos kaikki tiedostot täytyy tallentaa uudelleen. Tästä syystä panostin testaamiseen sen verran, että ainakaan tavallisimmissa toiminnoissa systeemi ei hajoaisi käsiin, eivätkä esimerkiksi erikoiset tiedostonimet hajottaisi järjestelmää.

Omat haasteensa asetti myös näyttökoneissa hyödynnettävä vanha toisto-ohjelma, jonka hallinta saneli monet Turbanin tallennusratkaisut. Valitettavasti minulla ei tätä kirjoittaessa ole tarpeeksi kokemusta C-kielellä ohjelmoinnista, saatika työpöytäsovelluksista, jotta olisin voinut tehdä paremman toisto-ohjelman. Tämä voisikin olla mahdollisesti tulevaisuuden kehitysaskel, sillä (kuten aiemmin mainittu) nykyinen ohjelma käyttää pitkälti valmiita Windowsin komponentteja erilaisten medioiden toistamiseen, joten uuden soitto-ohjelman rakentaminen ei välttämättä vaatisi kovin korkean tason C-kielen ohjelmointiosaamista. Samalla olisi mahdollista siirtyä käyttämään MySQL-tietokantaa tietojen tallentamiseen.

Näin jälkeenpäin ajatellen olisi ollut hienoa, jos mediapalvelimen päivitys olisi ollut tiedossa jo alusta alkaen. Kuten Toteutus-osiossa mainitsin, olisin valinnut PHP:n ja tekstitiedostojen sijasta mieluummin tekniikoiksi Ruby On Railsin ja MySQL-tietokannan niiden helppouden takia.

Itse toteutukseen olisi ollut parempi soveltaa jotain ketterän kehityksen menetelmää, sen sijaan, että rakensin koko paketin alusta loppuun ja muutin asioita matkan varrella. Näin jokaisen ominaisuuden olisi voinut testata käyttäjillä ja toisaalta Turban olisi saatu käyttöön lähes välittömästi.

Käytin mielestäni myös yksityiskohtien nysväämiseen ja koodin optimoimiseen kohtuuttoman paljon aikaa. Vaikka tämä helpottaa Turbanin jatkokehitystä ja parantaa sekä selaintukea että vakautta, olisi mielestäni kuitenkin ollut tärkeämpää saada Turban nopeammin käyttöön, vaikka hieman purkalla kasassa pysyvänä.

Loppujen lopuksi projektin tavoitteet kuitenkin täyttyivät hyvin, vaikka aikataulu venyikin reilusti. Urbaanimedian hallinta -luvussa kuvatut kolme pääongelmaa saatiin ratkaistua hyvin. Nyt Urbaanimedioiden hallinta on puuduttavan käsityön sijasta helppoa ja yksinkertaista. Yleisimmät toimenpiteet, kuten sisällön lisääminen ja poistaminen, vievät maksimissaan muutamia minuutteja. Järjestelmän opettelu siirrettiin vanhemman työntekijän harteilta suoraan verkkoon, joten tähän menee myöskin selvästi vähemmän aikaa. Automatisoidun sisällön poistamisen myötä myös asiakkaat, eli sisällöntuottajat, ovat tyytyväisempiä, kun heidän ei tarvitse pyytää ADC:n väkeä poistamaan vanhentunutta materiaalia näytöiltä.

LÄHTEET

Fried, Jason 2004. Interface Design Tip: Find the Epicenter. [Verkkodokumentti] <http://www.37signals.com/svn/archives/000737.php> (luettu 14.3.2009).

Garrett, Jesse James 2003. The Elements of User Experience: User-Centered Design for the Web. Berkeley, California, New Riders Publishing.

Hodge, Sean 2008. PSDTuts+: 7 Principles of Effective Icon Design [Verkkodokumentti] <http://psd.tutsplus.com/articles/7-principles-of-effective-icon-design/> (luettu 10.7.2010).

Krug, Steve 2006. Don't Make Me Think. A Common Sense Approach to Web Usability. Berkeley, California, New Riders Publishing.

Nielsen, Jakob 2000. Designing Web Usability. Berkeley, California, New Riders Publishing.

Nielsen, Jakob 2008. Top 10 Application Design Mistakes [Verkkodokumentti] <http://www.useit.com/alertbox/application-mistakes.html> (luettu 7.3.2010).

Nielsen, Jakob 2010. Scrolling and Attention [Verkkodokumentti] <http://www.useit.com/alertbox/scrolling-attention.html> (luettu 7.3.2010).

Parkkinen, Jarmo 2002. Hyvään verkkopalveluun! Tampere, Inforviestintä Oy.

Samela, Juha 2002. Verkkosisällön hallinta. Helsinki, Edita Publishing Oy.

Sinkkonen, Irmeli; Kuoppala, Hannu; Parkkinen, Jarmo; Vastamäki Raino 2006. Käytettävyyden psykologia. Helsinki, Edita Publishing Oy.

Swartzfager, Brian 2008. Problem: No Drag-and-Drop on the iPhone/iPod Touch. Solution: Click-to-click Move [Verkkodokumentti] <http://www.thoughtdelimited.org/thoughts/post.cfm/problem-no-drag-and-drop-on-the-iphone-ipod-touch-solution-click-to-click-move> (luettu 29.4.2010).

Wikipedia 2010. Affordance [Verkkodokumentti]
<http://en.wikipedia.org/wiki/Affordance> (luettu 29.4.2010).

Wikipedia 2009. AJAX. [Verkkodokumentti]
<http://en.wikipedia.org/wiki/Ajax_(programming)> (luettu 23.2.2009).

Wikipedia 2010. Drag-and-Drop [Verkkodokumentti]
<http://en.wikipedia.org/wiki/Drag_and_drop> (luettu 29.4.2010).

Wikipedia 2009. Web application [Verkkodokumentti]
<http://en.wikipedia.org/wiki/Web_application> (luettu 23.2.2009).

Yazdi, Fiz ja Leech, Joe 2009. The myth of the page fold: evidence from user testing [Verkkodokumentti]
<http://www.cxpartners.co.uk/thoughts/the_myth_of_the_page_fold_evidence_from_user_testing.htm> (luettu 10.7.2010).